Optimization of a Fast Transform Structured as a Convolutional Tree

Olivier Chabiron*[1], François Malgouyres[2], Herwig Wendt[1] and Jean-Yves Tourneret[1]

[1]Institut de Recherche en Informatique de Toulouse, IRIT-CNRS UMR 5505, ENSEEIHT, Toulouse, France
[2]Institut de Mathématiques de Toulouse, IMT-CNRS UMR 5219, Université de Toulouse, Toulouse, France

January 19, 2016

Abstract

To reduce the dimension of large datasets, it is common to express each vector of this dataset using few atoms of a redundant dictionary. In order to select these atoms, many models and algorithms have been proposed, leading to state-of-the-art performances in many machine learning, signal and image processing applications. The classical sparsifying algorithms compute at each iteration matrix-vector multiplications where the matrix contains the atoms of the dictionary. As a consequence, the numerical complexity of the sparsifying algorithm is always proportional to the numerical complexity of the matrix-vector multiplication. In some applications, the matrix-vector multiplications can be computed using handcrafted fast transforms (such as the Fourier or the wavelet transforms). However, the complexity of the matrix-vector multiplications very often limits the capacities of the sparsifying algorithms. It is particularly the case when the transform is learned from the data. In order to avoid this limitation, we study a strategy to optimize convolutions of sparse kernels living on the edges of a tree. These convolutions define a fast transform (algorithmically similar to a fast wavelet transform) that can approximate atoms prescribed by the user. The optimization problem associated with the learning of these fast transforms is smooth but can be strongly non-convex. We propose in this paper a proximal alternating linearized minimization algorithm (PALMTREE) allowing Curvelet or Wavelet-packet transforms to be approximated with excellent performance. Empirically, the profile of the objective function associated with this optimization problem has a small number of critical values with large watershed. This confirms that the resulting fast transforms can be optimized efficiently, which opens many theoretical and applicative perspectives.

1 Introduction

1.1 Sparse coding needs fast transforms

Modern methods for efficiently representing data such as signals and images express the data as sums of elementary elements called atoms. These atoms are gathered in a matrix denoted as $\mathbf{D}$ in this paper. Various models and algorithms have been proposed in the literature to estimate these atoms from the observed data. Many of these methods are based on optimization strategies involving non-differentiable objective functions. Among commonly used non-differentiable terms, the most popular convex ones are probably

*Olivier Chabiron is supported by ANR-11-LABX-0040-CIMI within the program ANR-11-IDEX-0002-02.

the ℓ^1 norm [Tib96, CDS98] and the grouped ℓ^1 norms [TVW05] that have been intensively studied in the literature (see [BJMO12] for a review). Also, non convex variants (e.g., the iterative hard thresholding [BD09]) have been investigated. However, even the most efficient of these algorithms requires to compute at each iteration matrix-vector multiplications with the matrix $\mathbf{D}$. Other popular strategies, often referred to as "greedy algorithms", iteratively estimate the set of selected atoms. Examples of such strategies involve the orthogonal matching pursuit (OMP) [PRK93] and variants such as COSAMP [NT09] (which is dedicated to incoherent dictionaries). Again, these strategies require to compute at each iteration matrix-vector multiplications with the matrix $\mathbf{D}$.

In order to use the methods mentioned above, it is therefore crucial to efficiently compute the matrix-vector multiplication involving $\mathbf{D}$. For unstructured dictionaries, it is possible to improve the algorithms computing these matrix-vector multiplications [Str69]. However, only limited improvements can be expected in general situations (see [Lan08], for a review on this subject). The only solution left is therefore to consider dictionaries whose structure permits to compute matrix-vector multiplications with some kind of fast transform. This might be difficult in machine learning applications where the dictionary contains the data. However, in signal and image processing applications, the dictionary contains simple bricks whose weighted sum approximates images. Not surprisingly, it is common to consider dictionaries made of hand-crafted atoms coming with some kind of fast transform. Famous examples include the discrete cosine transform, the fast wavelet transform (see [Mal99], for a review) or the curvelet transform [CDDY06]. However, the resulting dictionaries lack flexibility and may fail to efficiently represent some of the structures contained in the images. Thus these transforms are not able to reach state-of-the-art performance in many practical applications.

For the resolution of most inverse problems, better approximation performance is achieved when the observed data are decomposed using dictionaries learnt from the data themselves. The resulting dictionary learning (DL) problem has received increasing attention since the pioneering works of Lewicki, Sejnowski and Olshausen [LS00, OF97]. We invite the reader to consult [Ela10] for more details about sparse representations and DL. Typical DL models and algorithms rely on the resolution of non-convex and non-smooth high-dimensional optimization problems. Famous examples of such models/algorithms include the method of optimal directions [EAHH99] or the K-SVD algorithm [AEB06]. Distributed or online strategies enable more images to be used, limiting overfitting and avoiding local minima [MBPS10]. However, DL is usually applied to small image patches in order to avoid over-fitting and because of the computational cost induced by successive computations of matrix-vector multiplications involving $\mathbf{D}$. For unstructured dictionaries, this cost is indeed at least $O(N^2)$, where N is the sample/image size. Moreover, the cost of the dictionary update is usually at least $O(N^3)$. Learning fast transforms will, in this context, make it possible to estimate larger atoms for larger images.

1.2 The proposed fast transform

The fast transform structure considered in this paper is a convolutional tree depicted in Figure 1. A convolutional tree is defined from a rooted tree $\mathcal{T}(\mathcal{E},\mathcal{N})$ composed of nodes $\mathcal{N}$ and edges $\mathcal{E}$. Each edge connects two nodes. The root of the tree is denoted by r and the set containing all its leaves is denoted by $\mathcal{F}$. Given a positive integer S, to any edge $e \in \mathcal{E}$ is associated an S-sparse convolution kernel $h^e \in \mathbb{R}^N$, where N is the size of the lattice $\mathcal{P}$ of the signal/image of interest. Since there is a natural one-to-one correspondence between elements $i \in \{1,\dots,N\}$ and pixels $p \in \mathcal{P}$, for a kernel $h \in \mathbb{R}^N$, we equivalently write h_i or h_p, where p is the pixel in $\mathcal{P}$ corresponding to $i \in \{1,\dots,N\}$. The lattice is endowed with an additive commutative group structure corresponding to the natural addition of pixels when $\mathcal{P}$ is periodized. When it exists, we denote the unique path between the nodes n and $n' \in \mathcal{N}$ by $\mathcal{C}(n,n') = (e_1,\dots,e_l)$, where e_1 starts at n and e_l ends at n'. We denote the composition of the convolution kernels living on a

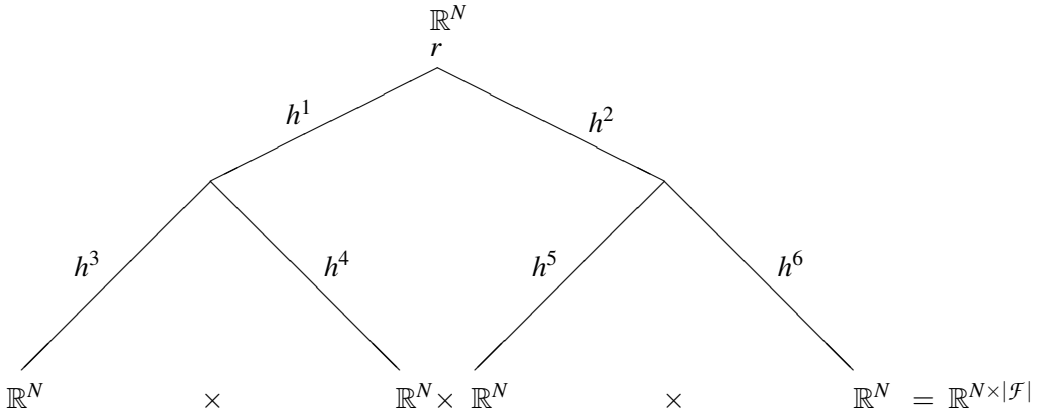


Figure 1: Example of convolutional tree considered in this paper. The resulting fast transform is defined by the tree and the S-sparse kernels $h^1, h^2, h^3, h^4, h^5, h^6$.

path $\mathcal{C}(n,n') = (e_1, \ldots, e_l)$ by

$$\mathbf{h}^{\mathcal{C}(n,n')} = h^{e_1} * \cdots * h^{e_l} \tag{1}$$

where we recall that for any h and $h' \in \mathbb{R}^N$, the circular convolution is defined by

$$(h * h')_p = \sum_{p' \in \mathcal{P}} h_{p-p'} h'_{p'} \qquad \forall p \in \mathcal{P}.$$

Given these notations, the considered dictionary contains all the translations of the compositions of convolutions

$$\mathbf{H}^f = \mathbf{h}^{\mathcal{C}(r,f)}, \qquad \text{where } f \in \mathcal{F} \text{ and } r \text{ is the root of } \mathcal{T}(\mathcal{E}, \mathcal{N}). \tag{2}$$

The code describing a data is $\mathbf{x} = (\mathbf{x}^f)_{f\in\mathcal{F}} \in \mathbb{R}^{N\times|\mathcal{F}|}$, where for every $f \in \mathcal{F}$, $\mathbf{x}^f \in \mathbb{R}^N$, we have

$$\mathbf{D}\mathbf{x} = \sum_{f\in\mathcal{F}} \mathbf{x}^f * \mathbf{h}^{\mathcal{C}(r,f)}. \tag{3}$$

It is not difficult to see that an implementation crossing the tree in a depth-first search (DFS) manner computes $\mathbf{D}\mathbf{x}$ with the computational complexity $O(|\mathcal{E}| \cdot S \cdot N)$. Moreover, denoting $\tilde{h}_p = h_{-p}$ (all the signals/images are assumed to be periodic), we can easily check that for any $u \in \mathbb{R}^N$

$$\mathbf{D}^T u = \left(u * \widetilde{\mathbf{h}^{\mathcal{C}(r,f)}} \right)_{f\in\mathcal{F}} \in \mathbb{R}^{N\times|\mathcal{F}|}.$$

Thus, an implementation crossing the tree in a breadth-first search (BFS) manner and using the property that

$$\widetilde{\mathbf{h}^{\mathcal{C}(r,f)}} = \tilde{h}^{e_1} * \cdots * \tilde{h}^{e_l}$$

computes $\mathbf{D}^T u$ with the computational complexity $O(|\mathcal{E}| \cdot S \cdot N)$.

The numerical complexity of the matrix multiplication involving $\mathbf{D}$ and its transpose is $O(|\mathcal{E}| \cdot S \cdot N)$, which is linear with the size of the signal/image. Therefore, such a transform can be applied to large signal/images. Note also that (surprisingly) the numerical complexity for computing matrix multiplications involving $\mathbf{D}$ does not depend of the number of columns in $\mathbf{D}$. Moreover, the computation of the convolutions can easily and efficiently be parallelized for many hardware architectures (including GPUs).

Interestingly, optimized implementations of similar transforms have been investigated for deep-learning using convolution neural networks architectures (among other, see [LKF10]).

Examples involving similar fast transforms (defined by convolutional trees) include the translation invariant wavelet and wavelet packet transforms. However, in the present transform the kernels $(h^e)_{e\in\mathcal{E}}$ can be different at any edge and can be optimized in order to approximate target atoms $(\mathbf{A}^f)_{f\in\mathcal{F}} \in \mathbb{R}^{\mathcal{P}\times\mathcal{F}}$. This optimization was discussed in [CMTD14] in the situation where the tree is restricted to a single branch (hence, single atom and its translations). The main contribution of this paper is to formulate, study and solve the problem for several target atoms.

1.3 Other works related to multi-layer/deep matrix factorization

Multi-layer (also called "deep") variational problems are currently receiving a growing attention for image representation. However, they have mostly been studied in the DL context. For instance, a method approximating a matrix by the product of two sparse matrices was proposed in [RZE10]. Dictionaries made of two layers based on a sparsifying transform and a sampling matrix were also investigated in [DCS09] (both layers are learnt by the algorithm). To the best of our knowledge, there only exist a few attempts for building dictionaries involving an arbitrary number of layers. In a slightly different context, dictionaries structured by Kronecker products have been proposed in [THZ13]. Interestingly, despite the non-convexity of the corresponding energy, it is possible to find some of its global minima [Wie12]. Dictionaries structured by wavelet-like trees (similar to the ones that we are targeting in this paper) using a dictionary update based on a gradient descent have been studied in [SO02]. A factorization of a matrix as a product of mutiple sparse matrices was also studied in [LMG14, LMGG15].

1.4 Contributions

In the present paper, we describe a model and build an algorithm for optimizing the fast transform we described in Section 1.2. We prove that the algorithm converges to a critical point of the problem and describe simple computational steps for its implementation. In most practical situations, the numerical complexity of one iteration of the algorithm is dominated[1] by the term $O\left(S^2.N.|\mathcal{E}|\right)$, where S is the size of the kernels. To illustrate the model and the algorithm, we consider experiments in which we target curvelet and wavelet packet dictionaries. Despite the constraint imposed by the fast transform structure, the approximation performances are excellent. We also illustrate empirically that the considered optimization problem apparently lends itself to global optimization.

1.5 Paper organization

Section 2 introduces our notations and describes the multi-layer matrix factorization problem whose optimization provides an optimized fast transform. Section 3 describes the PALMTREE algorithm solving this problem and presents a convergence statement. Section 4 studies several experiments in which a fast transform is optimized. In particular, some of these experiments suggest that the objective function can reasonably be globally optimized even when the depth of the tree is large (say equal to 10). Conclusions and some of the perspectives opened by this study are finally reported in Section 5.

2 Problem formulation

The image formation model studied in this paper considers an ideal dictionary $\mathbf{D}$ containing all the translations of a family of ideal unknown atoms $(\mathbf{A}^f)_{f\in\mathcal{F}}$. The image is formed by the synthesis operation $\mathbf{Dx}$

[1] A more detailed complexity bound is described in the paper.

which is the sum of convolutions between the known and simple codes $(\mathbf{x}^f)_{f\in\mathcal{F}}$ and the unknown target atoms $(\mathbf{A}^f)_{f\in\mathcal{F}}$. The observed data are, in this work, not corrupted by any noise leading to the following model

$$\mathbf{y} = \mathbf{D}\mathbf{x} = \sum_{f\in\mathcal{F}} \mathbf{x}^f * \mathbf{A}^f.$$

The fast transform optimization problem investigated in this paper consists of constructing a fast transform, structured as a convolutional tree, which approximates the matrix-vector multiplication with $\mathbf{D}$. More precisely, we want to find $(h^e)_{e\in\mathcal{E}}$ such that

$$\sum_{f\in\mathcal{F}} \mathbf{x}^f * \mathbf{A}^f \approx \sum_{f\in\mathcal{F}} \mathbf{x}^f * \mathbf{h}^{\mathcal{C}(r,f)} \tag{4}$$

where $\approx$ stands for "is well approximated by". The considered problem can also be seen as trying to factorize the matrix $\mathbf{D}$ as a product of matrices where the multiplications can be computed with a fast transform structured as a convolutional tree.

Typically, one might think of the target atoms $(\mathbf{A}^f)_{f\in\mathcal{F}}$ as handcrafted atoms such as curvelet or sinc atoms or other atoms with a prescribed time-frequency location. For simplicity, the codes $(\mathbf{x}^f)_{f\in\mathcal{F}}$ considered in this work are simple handcrafted translations of the Kronecker delta function, such that the atoms are well separated in $\mathbf{y}$. Given the constraints on the supports of $(h^e)_{e\in\mathcal{E}}$ (see below), the code is such that a good approximation in (4) guarantees good approximations of the target atoms

$$\mathbf{A}^f \approx \mathbf{h}^{\mathcal{C}(r,f)} \qquad \forall f \in \mathcal{F}.$$

As a consequence, an approximation of $\mathbf{D}$ is obtained since $\mathbf{D}\mathbf{x} = \sum_{f\in\mathcal{F}} \mathbf{x}^f * \mathbf{A}^f$.

The present framework might be understood as a dictionary update step of a DL algorithm. However, it significantly differs from a DL problem in which several input vectors $\mathbf{y}$ are available and the codes are estimated and sparse. Such a DL context is left for future work.

As already mentioned, the kernels $(h^e)_{e\in\mathcal{E}}$ are supposed to be S-sparse, which means that they contain a maximum of S non-zero elements. In the present work, we also assume that the supports of these kernels are fixed. Indeed, as shown in [CMTD14], many atoms (such as cosines, wavelets or curvelets) can be well approximated by compositions of convolutions of kernels with fixed supports.[2] In order to constrain the kernel supports, we define, for each edge $e \in \mathcal{E}$, an injective support mapping $\mathcal{S}^e \in \mathcal{P}^S$ which maps $\{1,\dots,S\}$ into the set of pixels $\mathcal{P}$. The range of a mapping $\mathcal{S}^e$ is

$$\mathrm{rg}\,(\mathcal{S}^e) = \{\mathcal{S}^e(1),\dots,\mathcal{S}^e(S)\} \subset \mathcal{P}.$$

We will impose in the sequel that, for any $e \in \mathcal{E}$, the support of the kernel h^e is included in this set, i.e.,

$$\mathrm{supp}\,(h^e) \subset \mathrm{rg}\,(\mathcal{S}^e).$$

Under this constraint, at every leaf $f \in \mathcal{F}$ the support of the composition of convolutions $\mathbf{h}^{\mathcal{C}(r,f)}$ is necessarily included in a set called the *reachable support*. It is not difficult to establish that this reachable support is defined for $f \in \mathcal{F}$ by

$$S^f = \left\{ p \in \mathcal{P} \,\middle|\, p = \sum_{e\in\mathcal{C}(r,f)} p_e, \text{ where } p_e \in \mathrm{rg}\,(\mathcal{S}^e), \forall e \in \mathcal{C}(r,f) \right\}. \tag{5}$$

In order to obtain a good approximation, it is of course crucial that, for all $f \in \mathcal{F}$, S^f allows one to capture most of the energy in $\mathbf{A}^f$.

[2] Finding a way to optimize the kernel supports is, of course, a natural extension of the present work.

Putting all this together, we obtain, for a known observed signal/image $\mathbf{y} \in \mathbb{R}^N$ and known codes $(\mathbf{x}^f)_{f\in\mathcal{F}} \in \mathbb{R}^{N\times|\mathcal{F}|}$, the following minimization problem

$$\text{(FTL)} \quad \begin{cases} \operatorname{argmin}_{(h^e)_{e\in\mathcal{E}}} \quad \left\|\sum_{f\in\mathcal{F}} \mathbf{x}^f * \mathbf{h}^{C(r,f)} - \mathbf{y}\right\|_2^2 \\ \text{under the constraint } h^e \in \mathcal{D}^e, \forall e \in \mathcal{E} \end{cases} \tag{6}$$

where the tree $\mathcal{T}(\mathcal{E},\mathcal{N})$ and the support mappings $(\mathcal{S}^e)_{e\in\mathcal{E}}$ are fixed, and for all $e \in \mathcal{E}$

$$\mathcal{D}^e = \left\{h^e \in \mathbb{R}^N |\ \operatorname{supp}(h^e) \subset \operatorname{rg}(\mathcal{S}^e),\ \|h^e\|_2 \leq \gamma\right\} \tag{7}$$

with a fixed parameter $\gamma > 0$.

The FTL problem has the following properties. The objective function of FTL is a polynomial whose variables are the reals h^e_p, with $e \in \mathcal{E}$ and $p \in \operatorname{rg}(\mathcal{S}^e)$. This polynomial is of degree $2K$, where K is the depth of $\mathcal{T}(\mathcal{E},\mathcal{N})$. It is infinitely differentiable and has a global Lipschitz gradient on any compact set. It is non-negative but might be non-convex. However, it is quadratic and convex with respect to any given kernel h^e, when the other kernels are fixed. For any $e \in \mathcal{E}$, the set $\mathcal{D}^e$ is closed and bounded and thus is a compact set. The set $\mathcal{D}^e$ is also convex. Finally, it is easy to show that the FTL problem has a minimizer.

Proposition 2.1 (Existence of a solution)**.**
For any tree $\mathcal{T}(\mathcal{E},\mathcal{N})$, any $(\mathbf{y},\mathbf{x},(\mathcal{S}^e)_{e\in\mathcal{E}}) \in \left(\mathbb{R}^N \times \mathbb{R}^{N\times|\mathcal{F}|} \times (\mathcal{P}^S)^{|\mathcal{E}|}\right)$ and any $\gamma > 0$ the FTL problem defined in (6) *has a minimizer.*

Indeed, the set $\mathcal{D}^e$ is compact for any $e \in \mathcal{E}$, where $\mathcal{E}$ is finite. Thus, the variables of our problem belong to a compact set. The objective function of (FTL) is continuous. Thus, the objective function is bounded on the compact set defined by the constraints and reaches its bounds. However, it is important to note that any global minimizer is not unique. For instance, it is always possible to multiply a kernel h^e by a non zero factor and to adjust the kernels in the subtree starting from the node ending at e in order to define the same dictionary $\mathbf{D}$.

3 Solving (FTL) with a proximal alternating linearized minimization

3.1 Proximal alternating linearized minimization

The *proximal alternating linearized minimization* (PALM) algorithm has been introduced in [BST14] to solve non-convex problems whose objective function is the sum of a smooth term and of non-smooth terms which only involve a block of the variables. The authors introduced their algorithm as a block-wise proximal forward-backward scheme. The applicability of the PALM algorithm has been investigated in [CPR13]. As the reader will see, this algorithm is perfectly adapted to the FTL problem. More precisely, the FTL problem satisfies the hypotheses guaranteeing the convergence of the PALM algorithm to one of its critical points. Moreover, any element of the resulting algorithm has a low computational complexity.

In order to describe the algorithm and to explain its properties, let us first formulate our problem as the problem studied in [BST14]. For this, we introduce, for any $e \in \mathcal{E}$, the following characteristic function

$$\chi_{\mathcal{D}^e}(h) = \begin{cases} 0 & \text{if } h \in \mathcal{D}^e, \\ +\infty & \text{otherwise,} \end{cases} \tag{8}$$

where $\mathcal{D}^e$ is defined in (7). We also rewrite the data fidelity term of the FTL problem using the notation

$$\Phi((h^e)_{e\in\mathcal{E}}) = \left\|\sum_{f\in\mathcal{F}} \mathbf{x}^f * \mathbf{h}^{C(r,f)} - \mathbf{y}\right\|_2^2. \tag{9}$$

Algorithm 1: Overview of the PALMTREE algorithm

Input:
$\mathbf{y}$: observed image;
$\mathcal{T}(\mathcal{E},\mathcal{N})$: tree defining the transform;
$(\mathbf{x}_f)_{f\in\mathcal{F}}$: codes;
$(\mathcal{S}^e)_{e\in\mathcal{E}}$: support mappings,
γ: parameter.
Output:
Kernels $(h^e)_{e\in\mathcal{E}}$.

begin
  Initialize h^e, for all $e\in\mathcal{E}$;
  while *not converged* **do**
    for $e'\in\mathcal{E}$ **do**
      Update $h^{e'} = \text{prox}_{t^{e'}}^{\chi_{\mathcal{D}^{e'}}}\left(h^{e'} - \frac{1}{t^{e'}}\nabla_{h^{e'}}\Phi((h^e)_{e\in\mathcal{E}})\right)$;

Using the notations introduced above, the FTL problem can be rewritten as follows

$$\text{argmin}_{(h^e)_{e\in\mathcal{E}}}\ \Phi((h^e)_{e\in\mathcal{E}}) + \sum_{e\in\mathcal{E}}\chi_{\mathcal{D}^e}(h^e). \tag{10}$$

The application of the PALM algorithm to solve (10) then takes the form described in Algorithm 1 (referred to as PALMTREE). We remind that the proximal operator is defined by

$$\text{prox}_t^{\chi_{\mathcal{D}^{e'}}}(h') = \underset{h\in\mathbb{R}^N}{\text{argmin}}\left\{\chi_{\mathcal{D}^{e'}}(h) + \frac{t}{2}\left\|h-h'\right\|_2^2\right\}. \tag{11}$$

In order to run PALMTREE, we need to compute three quantities

- the *proximal operator* associated with $\chi_{\mathcal{D}^{e'}}$,
- the *partial gradient* $\nabla_{h^{e'}}\Phi$ of the smooth part Φ of the objective function,
- the (inverse) stepsize $t^{e'}$: In the PALM algorithm, the (inverse) stepsize $t^{e'}$ must be strictly larger[3] than a *Lipschitz constant of the partial gradient* $h^{e'}\in\mathcal{D}^{e'}\longmapsto\nabla_{h^{e'}}\Phi((h^e)_{e\in\mathcal{E}})$, where the kernels $(h^e)_{e\neq e'}$ are fixed. We therefore need to compute such a Lipschitz constant.

We detail these computations in the upcoming Section 3.3. We will see that they are numerical tractable for numerous instances of the FTL problem. Note that, in PALMTREE, we can freely choose

- The initialization: Since we use the PALM algorithm to solve a non-convex problem, we might expect the initialization to be important. Surprisingly, we will see empirically that the initialization is not crucial since the profile of the objective function in (FTL) seems to have a large watershed leading to accurate approximations of y.
- The order to cross $\mathcal{E}$ in the "for" loop: From a numerical point of view, we will see that choosing a DFS order significantly decreases the computational complexity needed to compute $\nabla_{h^e}\Phi((h^e)_{e\in\mathcal{E}})$.

[3]We are aware of the fact that in [BST14] the condition on the stepsize is slightly stronger. We use this statement because the two conditions are equivalent when considering a finite number of iterations. This will always be the case for us in practice.

- The convergence criterion: Working on this stopping criterion provides an easy way to save time. For the numerical simulations reported below, we simply use a very large number of iterations that guarantees convergence.

Before explaining the computational details (see Section 3.3) and deriving the optimization algorithm (see Section 3.4), let us check that the sequence of kernels generated by the PALMTREE algorithm converges.

3.2 Convergence of PALMTREE

Proposition 3.1 (Convergence of PALMTREE)**.**
For any tree $\mathcal{T}(\mathcal{E},\mathcal{N})$, any $(\mathbf{y},\mathbf{x},(\mathcal{S}^e)_{e\in\mathcal{E}}) \in \left(\mathbb{R}^N \times \mathbb{R}^{N\times|\mathcal{F}|} \times (\mathcal{P}^S)^{|\mathcal{E}|}\right)$ and any $\gamma > 0$ the sequence generated by PALMTREE (see Algorithm 1) has finite length. It converges to a critical point of (FTL).

Proof. The convergence properties of PALMTREE are straightforward consequences of general results on the convergence of the PALM algorithm as described in the Theorem 1 of [BST14] (see also [CPR13]). Let us check that PALMTREE satisfies the hypotheses of this theorem.

1. The objective function of (FTL) satisfies the Kurdyka-Łojasiewicz property. In our case this is guaranteed (see [BST14]) by the fact the objective function of (FTL) is semi-algebraic.
2. For all $e \in \mathcal{E}$, the functions $\chi_{\mathcal{D}^e}$ are proper, non-negative and lower semi-continuous.
3. The function Φ is non-negative (since it is a squared norm) and continuously differentiable (since it is a polynomial). Moreover, its gradient is globally Lipschitz on the compact set defined by the constraints.
4. For any $e' \in \mathcal{E}$, any collection of kernels $h^e \in \mathcal{D}^e$, where $e \neq e'$, the partial gradient $h^{e'} \in \mathcal{D}^{e'} \longmapsto \nabla_{h^{e'}}\Phi((h^e)_{e\in\mathcal{E}})$ is globally Lipschitz. Moreover, since Φ is a polynomial, its Lipschitz moduli $L_{e'}((h^e)_{e\neq e'})$ are bounded away from $+\infty$ on the compact set defined by the constraints.
5. For all $e' \in \mathcal{E}$, as required in [BST14], we will set the stepsize $1/t^{e'}$ such that $t^{e'} > L_{e'}((h^e)_{e\neq e'})$, $L_{e'}((h^e)_{e\neq e'})$ is the Lipschitz moduli of the partial gradient.
6. The sequence generated by the algorithm is bounded since it belongs to the compact set defined by the constraints.

□

Note that we could obtain convergence rates for the PALMTREE algorithm by studying the parameters involved in the Kurdyka-Łojasiewicz property satisfied by our model.

3.3 Computation of the PALMTREE elements

3.3.1 The proximal operator

The proximal operator is defined by (11), where we remind that $\mathcal{D}^{e'}$ and its characteristic function are defined in (7) and (8). This proximal operator $\text{prox}_t^{\chi_{\mathcal{D}^{e'}}}(h')$ is therefore simply the projection of h' onto $\mathcal{D}^{e'}$. It is easy to check that it takes the following form

$$\text{prox}_t^{\chi_{\mathcal{D}^{e'}}}(h') = \frac{\gamma}{\max(\gamma, \|g^{e'}\|_2)} g^{e'} \tag{12}$$

where the projection $g^{e'}$ of $h' \in \mathbb{R}^N$ is defined by

$$g_p^{e'} = \begin{cases} h'_p & \text{, if } p \in \text{rg}\left(\mathcal{S}^{e'}\right), \\ 0 & \text{, otherwise.} \end{cases} \tag{13}$$

3.3.2 The partial gradient $\nabla_{h^{e'}}\Phi$

In order to compute the partial gradient of the smooth part Φ of the objective function, we consider an edge $e' \in \mathcal{E}$ denoted as

$$e' = (n, n') \qquad \text{where } n \text{ and } n' \in \mathcal{N}$$

and $(h^e)_{e\in\mathcal{E}} \in \mathbb{R}^{N\times|\mathcal{E}|}$. In order to compute $\nabla_{h^{e'}}\Phi((h^e)_{e\in\mathcal{E}})$, we need to isolate the block of variables $h^{e'}$ in the expression of the objective function and the quantities computed by our algorithm. For this, we need to introduce some new notations.

In order to discriminate the leaves of the tree terminating a branch that contains the edge e' from those that do not, we define the following subsets of $\mathcal{F}$

$$\mathcal{F}^{e'} = \{f \in \mathcal{F} | e' \in \mathcal{C}(r, f)\}$$
$$\overline{\mathcal{F}^{e'}} = \mathcal{F} \setminus \mathcal{F}^{e'}.$$

Given this notation, we define

$$\mathbf{y}^{e'} = \mathbf{y} - \sum_{f\in\overline{\mathcal{F}^{e'}}} \mathbf{x}_f * \mathbf{h}^{\mathcal{C}(r,f)}. \tag{14}$$

The composition of convolutions *above* the node n is denoted by

$$A^n = \mathbf{h}^{\mathcal{C}(r,n)}$$

whereas the sum of the convolutions *below* the node n' is denoted by

$$B^{n'} = \sum_{f\in\mathcal{F}^{e'}} \mathbf{x}_f * \mathbf{h}^{\mathcal{C}(n',f)}.$$

Finally, we define

$$H^{e'} = A^n * B^{n'} \tag{15}$$

satisfying the following relation

$$h^{e'} * H^{e'} = \sum_{f\in\mathcal{F}^{e'}} \mathbf{x}_f * \mathbf{h}^{\mathcal{C}(r,f)}.$$

The residual R is finally defined as

$$R = \sum_{f\in\mathcal{F}} \mathbf{x}_f * \mathbf{h}^{\mathcal{C}(r,f)} - \mathbf{y}. \tag{16}$$

Given the above notations, it is easy to check that

$$h^{e'} * H^{e'} - \mathbf{y}^{e'} = \sum_{f\in\mathcal{F}} \mathbf{x}_f * \mathbf{h}^{\mathcal{C}(r,f)} - \mathbf{y} = R.$$

Therefore,

$$\Phi((h^e)_{e\in\mathcal{E}}) = \|h^{e'} * H^{e'} - \mathbf{y}^{e'}\|_2^2$$

where $H^{e'}$ and $y^{e'}$ do not depend on $h^{e'}$. The following results are then obtained

$$\begin{aligned} \nabla_{h^{e'}}\Phi((h^e)_{e\in\mathcal{E}}) &= 2\,\widetilde{H^{e'}} * (h^{e'} * H^{e'} - \mathbf{y}^{e'}) && (17)\\ &= 2\,\widetilde{H^{e'}} * R \\ &= 2\,\widetilde{A^n} * \widetilde{B^{n'}} * R. && (18) \end{aligned}$$

In our tree data structure, in order to decrease the computational complexity associated with the evaluation of the gradient $\nabla_{h^{e'}}\Phi((h^e)_{e\in\mathcal{E}})$, we maintain variables representing A^n and B^n at every node $n \in \mathcal{N}$. A variable corresponding to the residual R is also defined. More precisely, these variables are defined as follows

- All the variables $(A^n)_{n\in\mathcal{N}}$ are pre-computed before the kernels $(h^e)_{e\in\mathcal{E}}$ are updated (i.e., before entering the "for" loop of Algorithm 1). This pre-computation is simply performed by initializing the variable A^r (at the root of the tree) with a Kronecker delta function and by computing the other variable A^n with a BFS order. As a consequence, when updating a node $n' \in \mathcal{N}$, the node n above it (i.e., such that $(n,n') \in \mathcal{E}$) has been updated. Therefore, the following convolution can be computed easily
$$A^{n'} = h^{(n,n')} * A^n. \tag{19}$$
Note that this computation is achieved with numerical complexity upper bounded by $O(S\cdot N)$. The update of all the variables $(A^n)_{n\in\mathcal{N}}$ is therefore pre-computed with a numerical complexity $O(S\cdot N\cdot|\mathcal{E}|)$.

- The variables $B^{n'}$ are updated in the "for" loop of Algorithm 1 before updating $h^{e'}$.

- The residual R is updated right after the update of $h^{e'}$.

It is important to recall here that in the "for" loop, we cross the edges of the tree with a DFS manner. Therefore, at the beginning of the block of instructions that updates $B^{n'}$, $h^{e'}$ and R, for $e' = (n,n') \in \mathcal{E}$ (i.e., the block of instructions of the "for" loop of Algorithm 1), we can assume that

- all the kernels on the edges below e' have been updated and therefore either $n' \in \mathcal{F}$ or, for any $n'' \in \mathcal{N}$ with $(n',n'') \in \mathcal{E}$, the variable $B^{n''}$ and kernels $h^{(n',n'')}$ are correct;

- the residual R is correct;

- none of the nodes and edges above e' has been updated. Therefore, none of the h^e, for e above e', has been updated and the pre-computed A^n is correct.

Denoting $\omega(n') = \{n'' \in \mathcal{N} | (n',n'') \in \mathcal{E}\}$, we update $B^{n'}$ according to
$$B^{n'} = \begin{cases} \mathbf{x}_{n'} & \text{, if } n' \in \mathcal{F} \\ \sum_{n''\in\omega(n')} h^{(n',n'')} * B^{n''} & \text{, if } n' \notin \mathcal{F}. \end{cases} \tag{20}$$

In order to update all the variables $(B^{n'})_{n'\in\mathcal{N}}$, we notice that the convolution with h^e is only performed once, whatever $e \in \mathcal{E}$. Therefore, the numerical cost for this update is $O(S\cdot N\cdot|\mathcal{E}|)$.

At this point of the algorithm, A^n, $B^{n'}$ and R are correct. We have therefore straightforward means (see (12), (18) and Algorithm 1) to compute $\nabla_{h^{e'}}\Phi((h^e)_{e\in\mathcal{E}})$, $t^{e'}$ and update $h^{e'}$. For clarity, we separate the description of these updates and describe them in Sections 3.3.3 and Section 3.3.4. Finally, in order to maintain R after the kernel $h^{e'}$ has been updated, the following update is required
$$R_{new} = R_{old} + (h^{e'}_{new} - h^{e'}_{old}) * B^{n'}. \tag{21}$$
This update is conducted with a numerical complexity of $O(S\cdot N)$ for any edge of the tree.

3.3.3 Updating $h^{e'}$

In order to update $h^{e'}$, for $e' = (n,n')$, assuming A^n, $B^{n'}$ and R are correct, we successively apply a gradient step and a proximal operator. According to Eqs. (12) and (13), the proximal operator reduces to a projection. As a consequence, computing $\nabla_{h^{e'}}\Phi((h^e)_{e\in\mathcal{E}})$ for pixels p located outside of $\mathrm{rg}\left(\mathcal{S}^{e'}\right)$ is useless. Therefore, in order to update $h^{e'}$, instead of computing the gradient $\nabla_{h^{e'}}\Phi((h^e)_{e\in\mathcal{E}})$ we only need to compute
$$\left(\nabla_{h^{e'}}\Phi((h^e)_{e\in\mathcal{E}})\right)_{\mathcal{S}^{e'}(s)} = (\widetilde{H^{e'}} * R)_{\mathcal{S}^{e'}(s)}, \qquad \text{for any } s \in \{1,\dots,S\},$$

where we remind that

$$H^{e'} = A^n * B^{n'}.$$

This step requires to compute $H^{e'}$. Depending on the situation, the cost for this computation can be optimized in many ways. However, a simple implementation based on the fast Fourier transform can always be used with a numerical complexity $O(N \cdot \log N)$. Note that when N is very large and the depth of the tree is small, it might be preferable not to use A^n and to recompute all the convolutions with the kernels above n. Once $H^{e'}$ has been computed, the numerical complexity for computing $\left(\nabla_{h^{e'}}\Phi((h^e)_{e\in\mathcal{E}})\right)_{\mathcal{S}^{e'}(s)}$, for $s \in \{1,\dots,S\}$ is $O(S \cdot N)$.

Another interesting point is that the useful entries of $\nabla_{h^{e'}}\Phi((h^e)_{e\in\mathcal{E}})$ can be written using matrix notations. If we denote by τ the usual translation operator defined by

$$(\tau_{p'} H^{e'})_p = H^{e'}_{p-p'},$$

and by $C_{e'}$ the matrix of size $N \times S$, whose sth column contains $\tau_{\mathcal{S}^{e'}(s)} H^{e'}$, the following result can be obtained

$$\left(\nabla_{h^{e'}}\Phi((h^e)_{e\in\mathcal{E}})\right)_{\mathcal{S}^{e'}(s)} = (C_{e'}^T R)_s \qquad \text{, for all } s \in \{1,\dots,S\}.$$

Finally, by using (12), the kernel $h^{e'}$ can be updated as follows

$$h^{e'}_{new} = \frac{\gamma}{\max(\gamma, \|g^{e'}\|_2)} g^{e'} \tag{22}$$

with

$$g^{e'}_p = \begin{cases} h^{e'}_p - \frac{1}{t^{e'}}(C_{e'}^T R)_s & \text{if there exists } s \text{ such that } p = \mathcal{S}^{e'}(s), \\ 0 & \text{otherwise,} \end{cases} \tag{23}$$

where we remind that we must have $t^{e'} > L_{e'}((h^e)_{e\neq e'}))$. The latter quantity is computed in the next section.

The complexity for updating $h^{e'}$ is essentially the complexity for computing $C_{e'}^T R$, which is $O(S \cdot N)$ for any edge of the tree.

3.3.4 Computing the Lipschitz constants $L_{e'}((h^e)_{e\neq e'}))$

Since $\mathcal{D}^{e'}$ is included in the set $\{h \in \mathbb{R}^N | \forall p \notin \text{rg}\left(\mathcal{S}^{e'}\right), h_p = 0\}$, which is an Euclidean space of dimension S, we are only interested in computing the Lipschitz constant of the gradient of the restriction of Φ to this space[4]. From (17) and using the matrix $C_{e'}$ defined in the previous section, this gradient can be expressed as

$$2C_{e'}^T(C_{e'} h^{e'}_{|\text{rg}(\mathcal{S}^{e'})} - \mathbf{y}^{e'}),$$

where $h^e_{|\text{rg}(\mathcal{S}^{e'})}$ is the restriction of $h^{e'}$ to its entries in $\text{rg}\left(\mathcal{S}^{e'}\right)$. Therefore, the Lipschitz constant $L_{e'}((h^e)_{e\neq e'}))$ must, for any h_1 and $h_2 \in \mathbb{R}^S$, verify

$$\left\|2C_{e'}^T C_{e'}(h_1 - h_2)\right\|_2 \leq L_{e'}((h^e)_{e\neq e'}))\, \|h_1 - h_2\|_2 .$$

Thus, we can take any Lipschitz constant such that

$$L_{e'}((h^e)_{e\neq e'})) \geq 2\, \rho(C_{e'}^T C_{e'}) \tag{24}$$

where $\rho(C_{e'}^T C_{e'})$ is the spectral radius of the square matrix $C_{e'}^T C_{e'}$, i.e., its largest eigenvalue. Considering the small size of this matrix (it is of size $S \times S$), computing the spectral radius has a reasonable cost, which is $O(S^3)$ for every edge of the tree. The stepsize $t^{e'}$ of the gradient descent is then adjusted in order to ensure $t^{e'} > L_{e'}((h^e)_{e\neq e'}))$.

[4]The convergence statements in [CPR13] hold for any (non empty) constraint. This is however not needed here since we only use the fact that our constraint is included in an Euclidean space (which is the case considered in [BST14]).

3.4 The detailed PALMTREE Algorithm

Finally, combining all the elements described in the previous sections, we obtain the PALMTREE algorithm described in Algorithm 2. For any iteration of the "while" loop, the computational cost of updating all kernels of the tree is dominated by

$$O\big(\ (S^3 + S^2 N + N\log(N))\ |\mathcal{E}|\big)\,.$$

Note that in the targeted practical applications, S is very small compared to N, so that the term S^3 is dominated by the other terms. Importantly, the computational cost scales pseudo-linearly with the number of image pixels. This property is very important and means that the PALMTREE algorithm can be used to optimize atoms living in large images. Another important point is that the number of leaves $|\mathcal{F}|$ has no influence on the algorithm computational cost. This cost scales linearly with the number of edges $|\mathcal{E}|$. For a fixed number of atoms in the dictionary, the more kernels are mutualised by the tree structure, the more efficient the dictionary update.

Algorithm 2: PALMTREE

Input:
$\mathbf{y}$: observed image;
$\mathcal{T}(\mathcal{E},\mathcal{N})$: tree defining the transform;
$(\mathbf{x}_f)_{f\in\mathcal{F}}$: codes;
$(\mathcal{S}^e)_{e\in\mathcal{E}}$: support mappings;
γ: parameter.
Output:
Kernels $(h^e)_{e\in\mathcal{E}}$.

begin
- Initialize kernels $(h^e)_{e\in\mathcal{E}}$;
- Initialize the residual R with (16); $O(S.N.|\mathcal{E}|)$
- **while** *not converged* **do**
  - **for** $n \in \mathcal{N}$ *(in BFS order)* **do**
    - Compute A^n with (19); $O(S.N.|\mathcal{E}|)$
  - **for** $e' = (n,n') \in \mathcal{E}$ *(in DFS order)* **do**
    - Update $B^{n'}$ with (20); $O(S.N.|\mathcal{E}|)$
    - Compute $H^{e'}$ with (15); $O(N\log(N).|\mathcal{E}|)$
    - Compute $C_{e'}^T C_{e'}$ where $C_{e'}$ defined in Section 3.3.3; $O(S^2.N.|\mathcal{E}|)$
    - Compute $L_{e'}$ with (24); $O(S^3.|\mathcal{E}|)$
    - Update $h^{e'}$ with (22) and (23); $O(S.N.|\mathcal{E}|)$
    - Update R with (21). $O(S.N.|\mathcal{E}|)$

4 Numerical experiments

Exploring all the possibilities of the proposed model cannot be achieved in a single paper. This is due to the fact that the choice of the tree $\mathcal{T}(\mathcal{E},\mathcal{N})$ and the support mappings $(\mathcal{S}^e)_{e\in\mathcal{E}}$ have a significant impact on the possibilities offered by this model. We have therefore deliberately restricted our attention to dictionaries

made of atoms with a precise localization both in the space domain and in the frequency domain. The motivation for considering this kind of dictionary is that they are ubiquitous in works related to image approximation and representation. More precisely, we consider and investigate the approximation of a dictionary of curvelets (see Section 4.2) and a dictionary of wavelet packets[5] (see Section 4.3). For these two dictionaries, we propose a generic way to construct $\mathcal{T}(\mathcal{E},\mathcal{N})$ and $(\mathcal{S}^e)_{e\in\mathcal{E}}$. This construction as well as the whole experimental setting are described in Section 4.1. We investigate how the atoms can be approximated both in the space and frequency domains by the proposed optimized fast transform. We also empirically study the profile of the objective function to be minimized when solving these problems (see Section 4.4). The purpose of this last study is to explore the critical values of the considered objective function and to assess the relative size of their convergence zones.

4.1 Experiment settings

4.1.1 Experimental protocol

The inputs of the PALMTREE algorithm described in Algorithm 2 are the data $\mathbf{y}$, the tree $\mathcal{T}(\mathcal{E},\mathcal{N})$, the codes $(\mathbf{x}_f)_{f\in\mathcal{F}}$, the support mappings $(\mathcal{S}^e)_{e\in\mathcal{E}}$ and the parameter γ. This section first describes the experimental protocol that we have considered to generate these inputs. In a second step, we introduce the quantities (denoted as PSNR*, SSG and NRE) that have been used to evaluate the results of the algorithm.[6] For all experiments, we consider a square image $\mathbf{y}$ of size $P=256$ such that $\mathcal{P}=\{0,\dots,255\}^2$ and $N=65536$.

Given a family of target atoms $(\mathbf{A}_f)_{f\in\mathcal{F}}$, each experiment aims at approximating a target dictionary $\mathbf{D}\in\mathbb{R}^{N\times(N|\mathcal{F}|)}$ whose columns are the translations of these target atoms. The target atoms $(\mathbf{A}^f)_{f\in\mathcal{F}}$ used in the experiments of Section 4.2 have been synthesized using the adjoint of the curvelet transform [CDDY06]. Conversely, in the experiments of Section 4.3, these atoms are apodized cosines generated from the inverse Fourier transform of the corresponding tile.

For every $f\in\mathcal{F}$, the code $\mathbf{x}^f\in\mathbb{R}^N$ contains a unique Kronecker delta function that can be located at different positions in the image $\mathbf{y}=\mathbf{D}\mathbf{x}$. However, these positions ensure that the translations of the target atoms are well separated from each other. The resulting images $\mathbf{y}$ used in the numerical experiments are displayed in Figure 3, when $\mathbf{D}$ contains curvelets, and in Figure 7, when $\mathbf{D}$ contains wavelet packets.

The parameter γ is set to an arbitrary large value. Indeed, we have observed that larger values of γ lead to better approximations (which is expected given the form of the (FTL) problem). Moreover, the proposed results are not very sensitive to the exact value of the parameter γ, when this parameter has been fixed to a large value.

For clarity, we separate the descriptions of the strategies used to construct the tree $\mathcal{T}(\mathcal{E},\mathcal{N})$ and to adjust the support mappings $(\mathcal{S}^e)_{e\in\mathcal{E}}$ in the next two sections.

4.1.2 From frequency tiling pyramids to convolutional trees

Again, the design of the tree is a subject in itself and its study is out of the scope of the present paper. However, to the best of our knowledge, it is the first time that a tree is built for the purpose considered in this article. The proposed construction is related to the frequency localization of the atoms. The principles leading to this construction is inspired by the well known connections between the wavelet and wavelet packet trees and the frequency localization of the corresponding bases [Mal99].

In order to build the tree, we first consider a frequency tiling pyramid. The considered tiling pyramids are displayed in Figure 2. The first tiling of the pyramid corresponds to the frequency localization of the

[5] Here, we abuse of the name "wavelet packet" because the atoms of the considered dictionary ressemble to the atoms of a complex wavelet packet frame (see [Kin01]). However, they are neither true wavelet packets nor true complex wavelet packets.

[6] Note that the evaluation is also based on visual inspection.

target atoms[7] $(\mathbf{A}^f)_{f\in\mathcal{F}}$. The tiling at an upper level is obtained by grouping some of the tiles of the tiling at the lower level. The tiling at the last level only contains a unique tile which covers the whole frequency plane. The tile obtained by grouping tiles at the lower level is referred to as the parent tile. The members of the group of tiles corresponding to a parent tile are termed the children tiles. In the experiments reported below, we will specifically consider the frequency tiling pyramids of the curvelet transform and of the complex wavelet packet transform. More precisely, the number of angular zones of a curvelet tiling will be initialized to 16 (32 with parity) in the outermost (high frequency) ring and is halved every two rings for inner rings. This tiling hence corresponds to a standard curvelet tiling [CDDY06]. Each level of the considered frequency tiling pyramid contains a curvelet tiling. The pyramid is illustrated in Figure 2.a (top). Every level of the wave packet frequency tiling pyramid is obtained by dividing the frequency plane into $(3^l)^2$ identical square parts, for some level $l \in \mathbb{N}$. The corresponding frequency tiling pyramid is illustrated in Figure 2 (a-bottom).

Given the frequency tiling pyramid and a fixed parameter $J \geq 1$, the tree is then built as follows:

- J ordered nodes are associated with each tile of each tiling of the pyramid
- These J nodes are connected by $J-1$ consecutive edges (according to the order of the nodes).
- The first of these nodes is connected to the last node of its parent tile in the tiling at the upper level.

The trees corresponding to the curvelet and the wavelet packet frequency tiling pyramids, obtained for $J = 1$ and 2, are represented in Figure 2. Given our construction, the tree $\mathcal{T}(\mathcal{E},\mathcal{N})$ is of depth $K = JL$, where L is the number of tilings in the pyramid. Therefore, increasing J enlarges the reachable support S^f (see (5)) for all $f \in \mathcal{F}$ and permits to improve the approximation of the target atoms. Each leaf $f \in \mathcal{F}$ corresponds to a tile of the tiling at the first level of the pyramid.

Note that the proposed construction ensures that (most of the time) *two atoms corresponding to close frequency tiles share many convolution kernels* h^e. Doing so, we balance the cost reduction for merging branches (rather than considering a number of disjoint branches equal to the number of atoms) with the diversity of atoms in the dictionary.

In the approximation experiments investigated in this study, we have considered $J = 2$ ordered nodes per tile. In the description of the profile of the objective function provided in Section 4.4, J varies between 1 and 3.

4.1.3 Kernel supports

We define the supports of the kernels according to the depth k of the corresponding edge in the tree[8]. More precisely, given the above parameter $J \geq 1$ and for a depth $k \in \{1,\dots,K\}$, for any $e = (n,n') \in \mathcal{E}$ such that n' is of depth k, we define the following kernel support

$$\mathrm{rg}\,(\mathcal{S}^e) = \lceil \max\left(1, \frac{k-1}{J}\right) \rceil \{-c,\dots,0,\dots,c\}^2 \tag{25}$$

where $\lceil . \rceil$ is the ceiling operator. In practice, these supports are concentrated for the root and the very next edge, then they are dilated and the dilation parameter increases every J levels. The closer to the leaves, the more dilated the supports. For example, for $c = 1$, $K = 4$ and $J = 1$, the range of the support mappings of depths 1 and 2 is $\{-1,0,1\}^2$, at the depth 3 is $\{-2,0,2\}^2$, and at the depth 4 is $\{-3,0,3\}^2$. Note that the centering of these supports at $p = (0,0)$ is ensured by the periodization of $\mathcal{P}$. Note also that the support size is $S = (2c+1)^2$.

[7] Notice that, by doing so, we implicitly assume that the frequency localization of the target atoms are well separated.

[8] We use the convention that the depth of an edge starting at the root is 1.

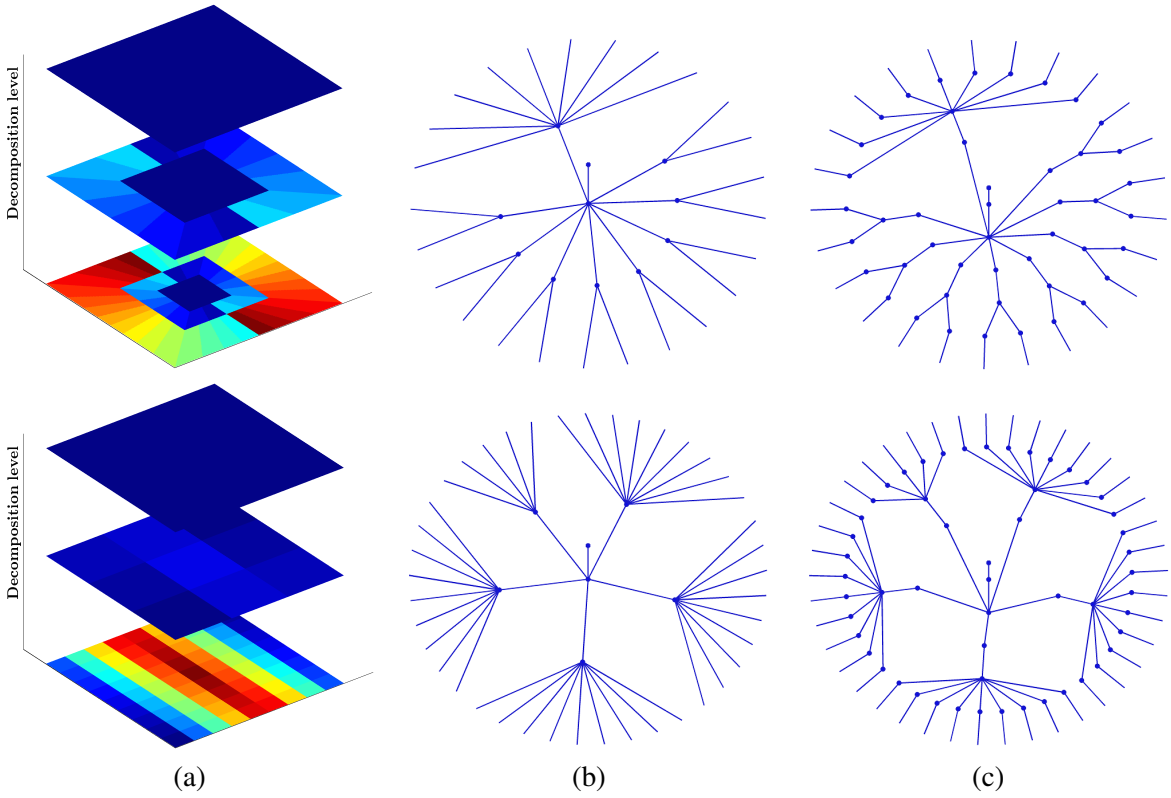


Figure 2: (a) Frequency tiling pyramid based on a level 3 curvelet decomposition (top) and wavelet packet decomposition (bottom). Corresponding trees with $J = 1$ (b) and $J = 2$ (c).

4.1.4 Performance evaluation

The atoms $(\mathbf{H}^f)_{f\in\mathcal{F}}$ provided by the proposed algorithm PALMTREE (defined in (2)) are referred to as optimized atoms. We propose to compare the optimized atoms with the target atoms $(\mathbf{A}_f)_{f\in\mathcal{F}}$ (used to build the data $\mathbf{y}$) using the following criteria.

PSNR*: The quality of the approximation of an atom $\mathbf{A}^f$ for a given $f \in \mathcal{F}$ with an optimized atom $\mathbf{H}^f$ is evaluated by the peak-signal-to-noise ratio (PSNR). More precisely, we consider a normalized PSNR denoted as PSNR*, which is normalized according to the size of the effective support of $\mathbf{A}^f$ in order to take into account that its support can be much smaller than $\mathcal{P}$. Defining the effective support of $\mathbf{A}^f$ as

$$\mathrm{supp}_{\mathrm{eff}}(\mathbf{A}^f) = \left\{p \in \mathcal{P} \mid |\mathbf{A}^f_p| \geq 10^{-2}(\max_{p\in\mathcal{P}}|\mathbf{A}^f_p|)\right\} \tag{26}$$

the normalized PSNR is obtained as follows

$$\mathrm{PSNR}^* = 10\,\log_{10}\left(\frac{r^2}{\mathrm{MSE}_{\mathrm{eff}}}\right)$$

where $r = \max_{p\in\mathcal{P}}(\mathbf{A}^f_p) - \min_{p\in\mathcal{P}}(\mathbf{A}^f_p)$ is the dynamic range of the atom $\mathbf{A}^f$ and

$$\mathrm{MSE}_{\mathrm{eff}} = \frac{\left\|\mathbf{H}^f - \mathbf{A}^f\right\|_2^2}{|\,\mathrm{supp}_{\mathrm{eff}}(\mathbf{A}^f)|} \tag{27}$$

is the normalized mean-squared error (MSE). Note that though PSNR* is a practical measure of the atom reconstruction quality, the optimization problem (6) aims at minimizing a data-fidelity term which involves several atoms. Therefore, we expect that the values of PSNR* will differ from one atom to another.

Search space gain (SSG): A compression gain denoted as SSG is defined as the ratio between the sum of the size of the effective supports of the target atoms (cf. (26)) and the search space size

$$\mathrm{SSG} = \frac{\sum_{f\in\mathcal{F}} \mathrm{supp}_{\mathrm{eff}}(\mathbf{A}^f)}{S|\mathcal{E}|}. \tag{28}$$

The larger SSG, the faster the transform (compared to the straightforward implementation of the convolutions).

Normalized reconstruction error (NRE): The normalized reconstruction error (NRE) provides information about the convergence of the optimization algorithm as well as about the approximation of the target atoms. It is defined by

$$\mathrm{NRE} = \frac{\|\mathbf{D}\mathbf{x}-\mathbf{y}\|_2^2}{\|\mathbf{y}\|_2^2} \tag{29}$$

and thus quantifies the data-fidelity term that the optimization problem (6) aims at minimizing.

4.2 Experiment targeting a curvelet dictionary

In this experiment, we target atoms from a dictionary of curvelets. The data $\mathbf{y}$, the codes $(\mathbf{x}_f)_{f\in\mathcal{F}}$ and the parameter γ have been generated as described in Section 4.1.1. The tree $\mathcal{T}(\mathcal{E},\mathcal{N})$ is constructed as described in Section 4.1.2 and the support mappings $(\mathcal{S}^e)_{e\in\mathcal{E}}$ are chosen as in Section 4.1.3. Two support parameters $c=1$ and $c=2$ are considered.

4.2.1 Search space gain

The tree defined in Section 4.1 is composed of 70 kernels corresponding to 630 and 1750 degrees of freedom (for $c=1$ and $c=2$). Storing all $|\mathcal{F}|=25$ target atoms would require 17358 values (the sum of their effective support sizes). Therefore, the search space gains are SSG ≈ 27.5 for $c=1$ and SSG ≈ 9.9 for $c=2$, which demonstrates the efficiency of the representation.

4.2.2 Approximation quality

Figure 3 compares the synthetic image with its reconstructions. Overall, the approximations are coherent with the target image, both for $c=1$ and $c=2$. The associated reconstruction errors are NRE $=0.2685$ for $c=1$ and NRE $=0.114$ for $c=2$. As expected, increasing the size of the kernel supports improves the

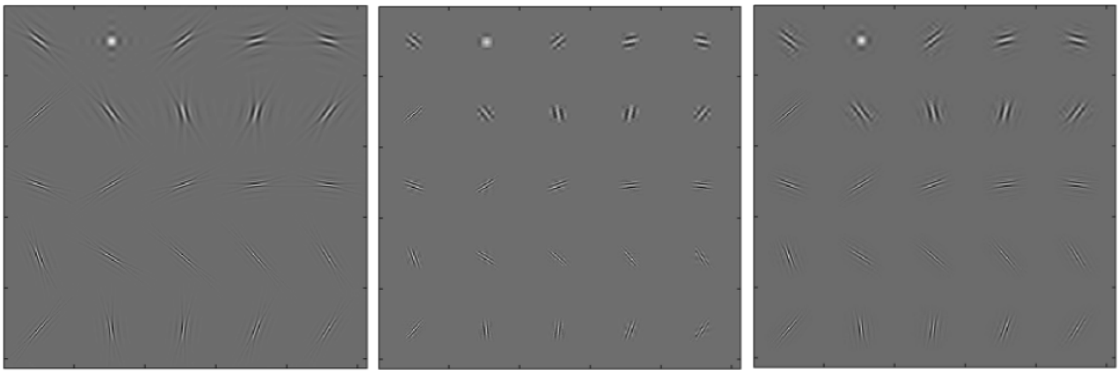


Figure 3: Curvelet dictionary approximation: (left) Synthetic data $\mathbf{y}=\sum_{f\in F}\mathbf{x}^f * \mathbf{A}^f$; (center) image $\sum_{f\in F}\mathbf{x}^f * \mathbf{H}^f$ obtained for $c=1$; (right) image $\sum_{f\in F}\mathbf{x}^f * \mathbf{H}^f$ obtained for $c=2$.

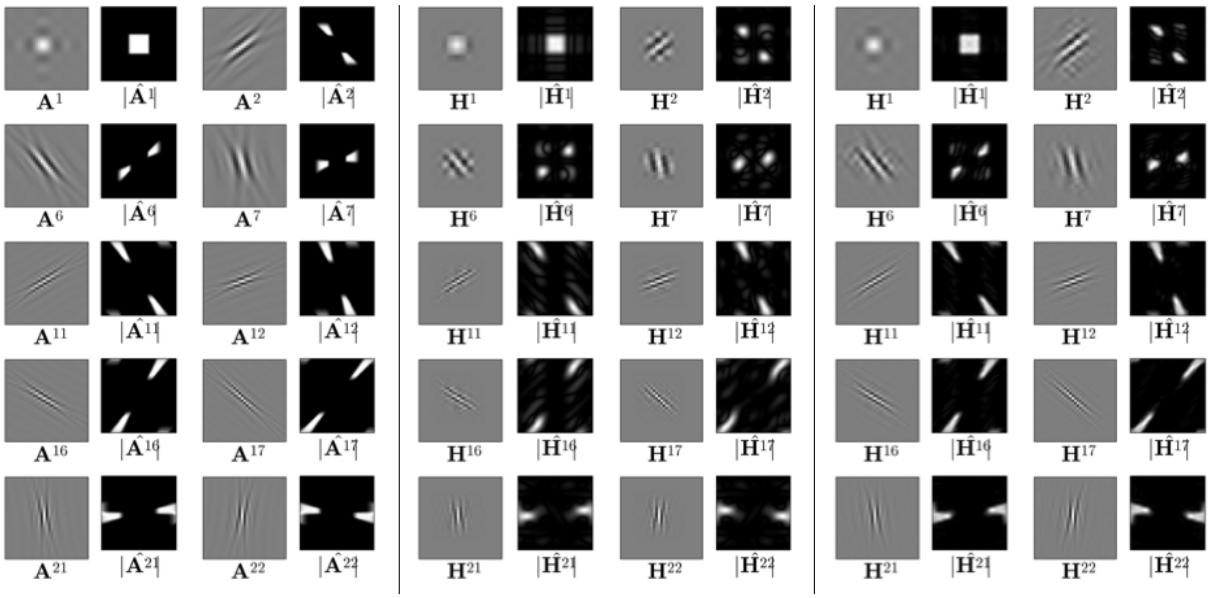


Figure 4: Comparison between curvelet atoms: Target atoms obtained by the inverse curvelet transform (a), optimized atoms with $c = 1$ (b) and with $c = 2$ (c). Note that the left columns correspond to the atoms whereas the right columns show the corresponding frequency responses.

PSNR*	$\mathbf{H}^1$	$\mathbf{H}^2$	$\mathbf{H}^6$	$\mathbf{H}^7$	$\mathbf{H}^{11}$	$\mathbf{H}^{12}$	$\mathbf{H}^{16}$	$\mathbf{H}^{17}$	$\mathbf{H}^{21}$	$\mathbf{H}^{22}$
$c = 1$	27	25.4	25.4	28.1	28.9	32.1	29	28.9	31.2	31.2
$c = 2$	35	28.9	28.9	33.9	38	37.8	38.1	37.2	38.8	38.9

Table 1: PSNR* of some optimized curvelet atoms, for 3×3 supports ($c = 1$) and 5×5 supports ($c = 2$).

approximation quality achieved by the optimized atoms, at the price though of increased computational cost and decreased compression capability. A more detailed visual comparison between target atoms and optimized atoms is shown in Figure 4 (left columns of subfigures (a) to (c)) displaying 10 of the 25 atoms of Figure 3. We observe that the optimized atoms are close to the target atoms. The size of the support has clearly a strong impact on the result, which is significantly better for $c = 2$ (support 5×5) than for $c = 1$ (support 3×3). In particular, with $c = 1$, we can observe that the tails of the target curvelets are not well reproduced by the optimized atoms. This difference is further quantified in Table 1, which provides the values of PSNR* for the atoms displayed in Figure 4. The improvement in terms of PSNR* when moving from $c = 1$ to $c = 2$ is substantial. However, the price to pay with this improvement is an increased computational cost (in particular, the computation complexities associated with the computations of $C_e^T C_e$ and L_e are of the orders $O(S^2)N$ and $O(S^3)$ respectively). For completeness, we note that the values of PSNR* obtained for $|\mathcal{F}| = 25$ atoms and $c = 1$ belong to the interval $[25.4, 32.1]$ with an average value equal to 29. For $c = 2$, the values of PSNR* belong to the interval $[28.9, 38.9]$ with an average value equal to 35.8. Finally, the frequency supports of the optimized atoms are compared in Figure 4 (right columns of subfigures (a) to (c)) with those of the target atoms. This comparison indicates that the convolutional tree used in this study is highly effective for recovering the frequency tiles of the target atoms, even if the choice of a small support $c = 1$ yields low-energy artifacts.

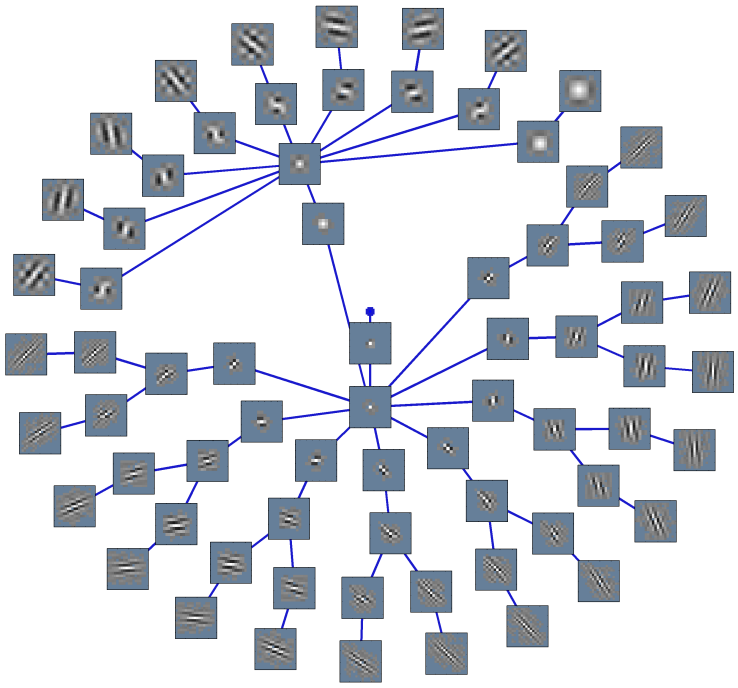


Figure 5: Decomposition of the action of the transpose operator $\mathbf{D}^T$ on a Kronecker function (dictionary optimized with $L = 3$, $J = 2$ and $c = 1$). Compositions of convolutions of the optimized kernels are computed along the branches of the tree. At the end of each edge $e = (n, n')$, we represent $\mathbf{h}^{C(r,n')}$. Thus, the optimized atoms are shown at the leaves of the tree. Images are zoomed toward the root to improve visibility.

4.2.3 Analysis of the optimized compositions of convolutions

We first illustrate the action of the transpose operator $\mathbf{D}^T$ on a Kronecker delta function. Figure 5 shows the successive convolutions of kernels (with inverted axes) from the root to the leaves for $J = 2$ and $c = 1$. We observe that despite the imposed nature of this six level structure, each kernel in the tree has learned a specific role for the approximation. In particular, note that different branches immediately lead to different shapes. Note also that the shapes formed at the 4th level of the tree clearly mimic curvelets from a level 2 decomposition, despite the fact that these shapes are neither involved in the model nor in the data. This phenomenon is further investigated in Figure 6, which shows these 4th level shapes and level 2 curvelets, together with their respective frequency responses. We can conclude that the shapes obtained at the 4th level are very similar to the level 2 curvelets although the algorithm has not been confronted with such atoms in the data.

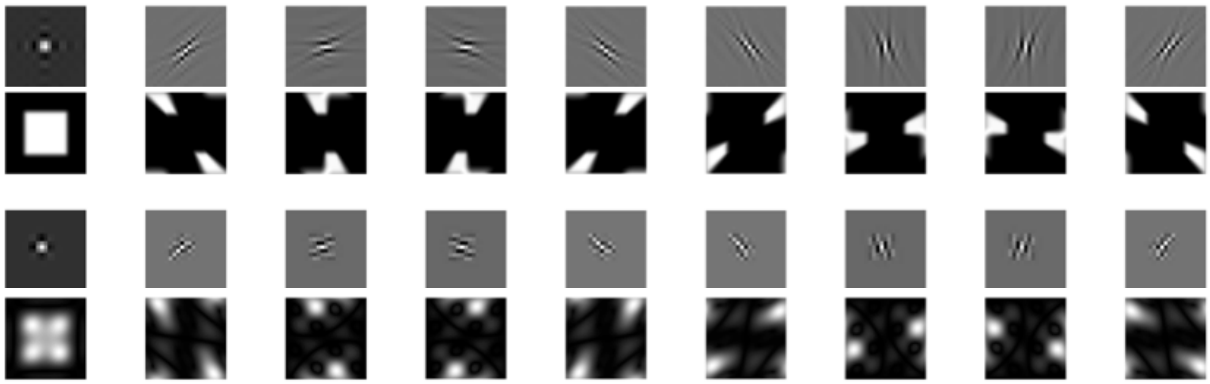


Figure 6: Actual curvelet atoms of a level $L = 2$ curvelet transform with their frequency content (top rows) and the corresponding shapes $\mathbf{h}^{C(r,n)}$, for the nodes n locate at the depth 4 of $\mathcal{T}(\mathcal{E},\mathcal{N})$, after optimizing a dictionary which level $L = 3$ curvelets and $c = 1$ (bottom rows).

4.3 Experiment targeting a Wavelet packet dictionary

We complement the numerical results presented in the previous section by an experiment trying to approximate atoms from a wavelet packet dictionary using a convolutional tree. The data $\mathbf{y}$, the codes $(\mathbf{x}_f)_{f\in\mathcal{F}}$ and the parameter γ are defined as in Section 4.1.1. The tree $\mathcal{T}(\mathcal{E},\mathcal{N})$ is constructed as described in Section 4.1.2 whereas the support mappings $(\mathcal{S}^e)_{e\in\mathcal{E}}$ are chosen as in Section 4.1.3. Again, two support parameters $c = 1$ and $c = 2$ are considered.

4.3.1 Search space gain

The tree is composed of 94 kernels, corresponding to 846 and 2350 degrees of freedom for $c = 1$ and $c = 2$. The sum of the effective support sizes of the $|\mathcal{F}| = 41$ target atoms is 13429. The proposed representation yields a search space gain of SSG ≈ 15.9 for $c = 1$ and SSG ≈ 5.7 for $c = 2$. The search space gain is smaller than for the previous experiment since the wavelet packet atoms have a more compact support. For this reason, kernel supports of size 5×5 ($c = 2$) are not appropriate for atom compression.

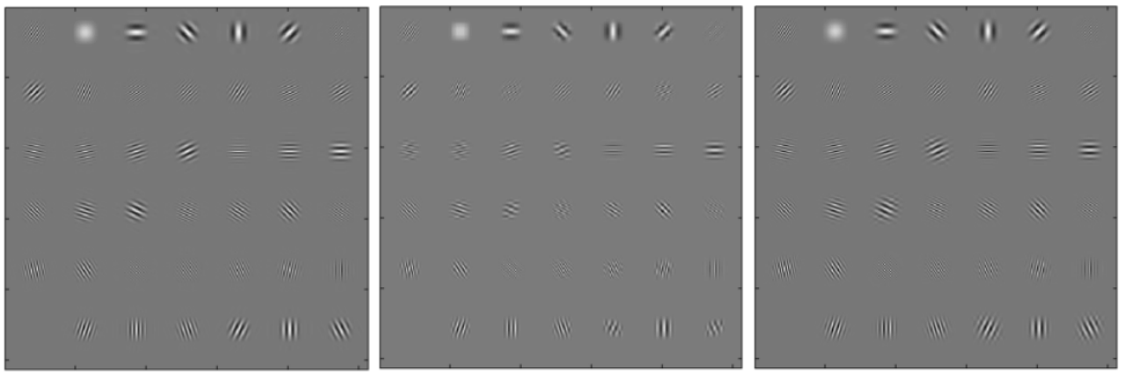


Figure 7: Wavelet packet dictionary approximation: (left) Synthetic data $\mathbf{y} = \sum_{f\in F} \mathbf{x}^f * \mathbf{A}^f$; (center) image $\sum_{f\in F} \mathbf{x}^f * \mathbf{H}^f$ for $c = 1$; (right) image $\sum_{f\in F} \mathbf{x}^f * \mathbf{H}^f$ for $c = 2$.

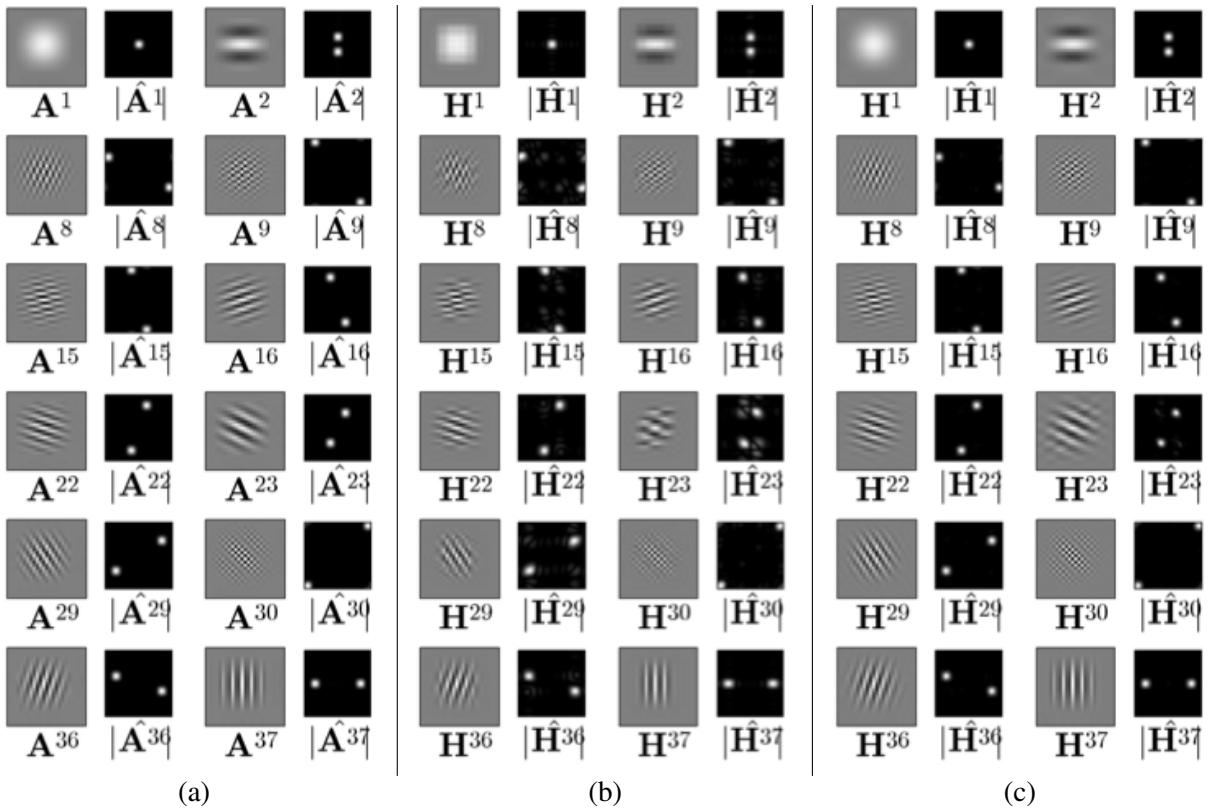


Figure 8: Comparison between wavelet packet atoms: Synthetic target atoms (a), optimized atoms with $c = 1$ (b) and $c = 2$ (c). The left columns of the subfigures show the atoms whereas the right columns correspond to the associated frequency responses.

4.3.2 Approximation quality

A visual comparison between the synthetic and the optimized atoms can be conducted using Figure 7. The associated reconstruction error is NRE $= 0.131$ for $c = 1$ (kernel support 3×3) and NRE $= 0.001$ for $c = 2$ (kernel support 5×5), indicating that the atoms with larger support are highly efficient for approximating the target atoms. This also means that the solution obtained by the algorithm is close (in terms of the value of the objective function) to a global minimum.

Figure 8 (left columns of subfigures (a) to (c)) provides a more detailed comparison for a selection of 12 out of the $|\mathcal{F}| = 41$ target atoms with their optimized counterparts. This figure suggests the following conclusions. First, the optimized atoms are very close to the actual atoms. Similarly to the curvelet atoms investigated in the previous section, the wavelet packet atoms are in most cases accurately approximated on their entire effective support for $c = 2$. For $c = 1$, the reachable support is limited. Note that some optimized atoms are less accurate than others, e.g., for $\mathbf{H}^{23}$ whose optimized atom seems to contain an additional weak energy contribution associated with a secondary frequency tile. This difference in accuracy can also be observed in Figure 8 (right columns of subfigures (a) to (c)), where the frequency supports of the optimized atoms are compared with those of the target atoms. In comparison with the results obtained with the curvelet dictionary approximation, we can observe that the optimized atoms have a better space-frequency localization, except that there are aliasing-like artifacts for some atoms such as, e.g., $\mathbf{H}^{23}$. Note that such artifacts are reduced when the support is increased from $c = 1$ to $c = 2$. The corresponding frequency tiles are those of slightly diagonal high frequency atoms. This indicates that the isotropic supports defined in (25) may not be well adapted to deal with these orientations at high frequency. Finally, Table

PSNR	$\mathbf{H}^1$	$\mathbf{H}^2$	$\mathbf{H}^8$	$\mathbf{H}^9$	$\mathbf{H}^{15}$	$\mathbf{H}^{16}$	$\mathbf{H}^{22}$	$\mathbf{H}^{23}$	$\mathbf{H}^{29}$	$\mathbf{H}^{30}$	$\mathbf{H}^{36}$	$\mathbf{H}^{37}$
$c=1$	25	29.9	23.6	28	24.8	28.6	28.6	20.1	23.4	29.3	28.6	26.9
$c=2$	41.6	46.8	39.9	44.3	41.8	43.3	43.1	29.4	42.4	44.2	43.3	43.9

Table 2: Values of PSNR* for some optimized wavelet packet atoms, for 3×3 supports ($c=1$) and 5×5 supports ($c=2$).

2 provides a quantitative summary of the approximation quality in terms of PSNR* values for the atoms displayed in Figure 8. The PSNR* values for all $|\mathcal{F}|=41$ atoms belong to the interval $[20.1, 32.3]$ with an average value equal to 26.3 (which is slightly below the one obtained with the curvelet atoms) for $c=1$. For $c=2$, the PSNR* values belong to the interval $[29.4, 46.8]$ with an average value equal to 41.7.

4.4 Profile of the objective function

The problem (FTL) is non-convex and convergence to a global minimizer is therefore not guaranteed. Of course, the value of a global minimum is unknown and is affected by the parameter settings (indeed, neither curvelet nor wavelet packet atoms have been generated by the convolutional tree, used as an input).

This section illustrates the good convergence properties of the PALMTREE algorithm. In order to do so, we provide features associated with the profile of the objective function. Although empirical, these features estimate the number of critical values[9] and their values, as well as the relative size of the corresponding watershed. This estimation is conducted for different values of the support size S and different values of J (see Sections 4.1.2 and 4.1.3). More precisely, with support sizes ranging from 2×2 to 5×5 and $J\in\{1,2,3\}$, we rerun 499 times the experiments of Sections 4.2 and 4.3 using random initializations obtained by independently drawing each $(h^e)_{e\in\mathcal{E}}$ from the uniform distribution in $\mathcal{D}^e$.

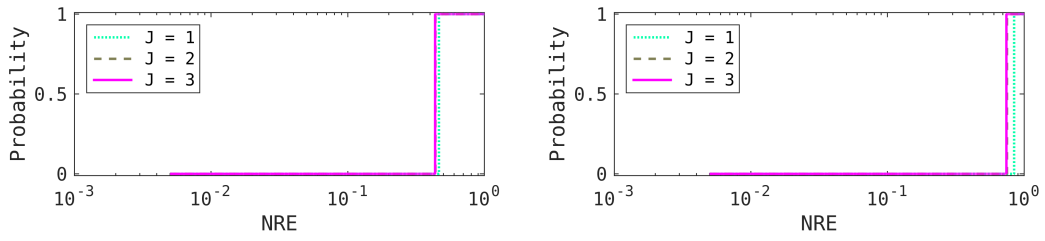
The profile of the objective function was assessed, for each set of parameters, by using the cumulative distribution function of NRE obtained at the last iteration of PALMTREE with the above random initialization. Intuitively, in case of perfect convergence to critical values, this cumulative distribution should be piecewise constant. Every critical value corresponds to a discontinuity of this piecewise constant function. Moreover, the relative size (in $\mathcal{D}^e$) of the subset of $\mathcal{D}^e$ leading to the critical value corresponds to the height of the jump associated with the discontinuity. When the convergence is not perfectly reached, the steps of the cumulative distribution function are blurry (this phenomenon is enhanced by the fact that we use a logarithmic scale for the abscissa of the curves presented in Figure 9).

The results are displayed in Figure 9 for the curvelet (left) and wavelet packet (right) atoms. The following three key conclusions can be obtained

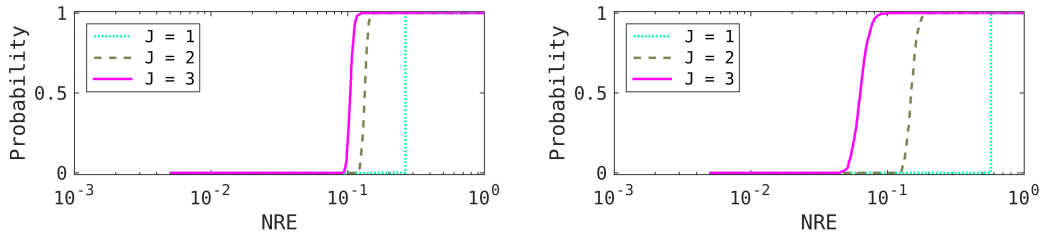
- The obtained values of NRE are always concentrated in a very narrow range, which tends to be larger when S and J are increased. In most cases, there actually seems to be only one single attraction pool (cf., Figure 9), up to the precision of our estimator. The NRE value is above (resp. below) a threshold with probability one (resp. zero). This property can be explained by the fact that the objective function has a unique critical value or at least, if another critical value exists, the watershed leading to it is small and not seen with the current convergence accuracy.

- The presence of smooth slopes (and a few staircases) for large degrees of freedom suggests an imperfect convergence in the same pool of attraction, potentially caused by an objective function that is flat around this minimizer. Finally, despite the stronger impact of non-convexity, an increased number of kernels or a larger support size helps to improve the NRE of the attained minimum. Indeed, the values of NRE that are obtained for large degrees of freedom with probability (estimated

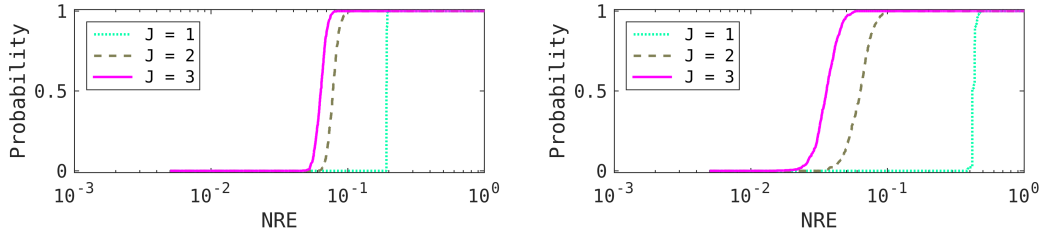
[9] A critical value is the value of the objective function at a critical point. To be precise we estimate the number of values that can be reached by PALMTREE.



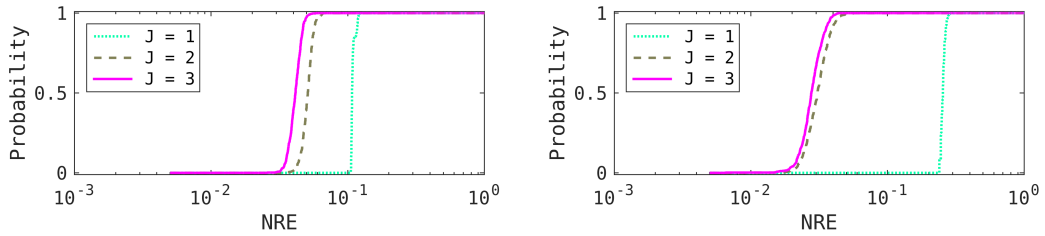
(a) For the support $S = 2 \times 2$



(b) For the support $S = 3 \times 3$



(c) For the support $S = 4 \times 4$



(d) For the support $S = 5 \times 5$

Figure 9: Estimated cumulative distribution function of NRE obtained for the curvelet (left) and wavelet packet (right) dictionary approximations, when kernels are initialized with random values.

to) one (Figure 9, bottom row) are consistently and significantly smaller than those obtained for less flexible kernels (Figure 9, upper rows).

- When the algorithm is run with a deterministic initialization defined by a centered Kronecker delta function (instead of the random initialization defined above), the resulting NRE is almost always equal to the lowest value of NRE obtained from the 499 random initializations. Of course, this result cannot guarantee that the solution is always localized in the pool of attraction of a good local minimizer for other atoms than those considered here. However, it indicates that the centered Kronecker delta function (the neutral element of the discrete convolution) is a sensible choice of initialization that can yield good results in situations where the average number of random restarts needed to obtain better NRE would be prohibitive.

5 Conclusions and perspectives

This paper proposed a model and an algorithm for optimizing a fast transform. The model builds on and extends the problem considered in [CMTD14] studying the capabilities of consecutive convolutions of sparse kernels to approximate a single atom of a given dictionary. More precisely, this paper considered the general problem of building a fast transform that optimizes an entire dictionary of atoms using S-sparse convolution kernels that are organized in a tree. The rationale of the tree structure is to arrange the convolution kernels in an intelligent way that maximizes the efficiency of the synthesis operator. Together with the sparsity of the kernels, this tree structure yields a compact representation of the elements of the dictionary. The numerical complexity for computing both the fast transform and its adjoint is $O(|\mathcal{E}|.S.N)$, where N is the number of pixels in the image, $\mathcal{E}$ denotes the edges of the tree and S is the sparsity of the kernels. In order to determine this structured fast transform, we defined a proximal alternating linearized minimization based algorithm that exploits the tree structure by adopting an efficient update order. One of the key aspects of the proposed model and algorithm is that, despite the fact that the model consists of many atoms, the complexity of each iteration is dominated[10]. by $O(S^2.N.|\mathcal{E}|)$.

In particular, it *scales linearly* with the image size. Furthermore, the structure of the algorithm is such that an update step for the code could be incorporated without specific modifications. This paper assumed a known and fixed code and focused on the dictionary update step. The full dictionary learning problem optimizing the code and the dictionary simultaneously will be the subject of a future work.

The approximation capabilities and convergence properties of the proposed fast transform model and algorithm were assessed and studied via several numerical experiments. The results indicate that the proposed model is effective for approximating large classes of atoms. Furthermore, although the problem is non-convex, our experiments show that the domain of attraction of the global minimum is either large (and it is realistic to expect that it can be found in practice) or very small (but the estimated singular points provide good approximations of the kernels). In either case, the proposed optimization algorithm PALMTREE was successful in finding a good approximation for the target atoms.

This paper considered a fixed tree structure that obviously determines the nature of atoms that can be well approximated. The tree structure used in the numerical experiments was inspired by the frequency tiling pyramids of multiresolution decomposition schemes. Other tree designs, as well as the possibility to partially learn the tree from the data, are currently being studied. Another promising perspective for increasing the flexibility of the model is to estimate the sizes and shapes of the supports of the convolution kernels from the observed data (whereas the sparse support was fixed in this study). This will also be considered in future work. Finally, to go further and learn a fast transform from a large number of images, it would be interesting to study an online or distributed version of the algorithm.

[10] A more precise expression is provided in the paper. In most standard configurations, it is dominated by the above term (e.g., when $S = 10$, to get $log(N) = S^2$ we need to take $N = e^{100} \sim 2.10^{43}$ which is unrealistic)

References

[AEB06] M. Aharon, M. Elad, and A. M. Bruckstein. The K-SVD, an algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Trans. Signal Process.*, 54(11):4311–4322, 2006.

[BD09] T. Blumensath and M. E. Davies. Iterative hard thresholding for compressed sensing. *Applied and Computational Harmonic Analysis*, 27(3):265–274, 2009.

[BJMO12] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends® in Machine Learning*, 4(1):1–106, 2012.

[BST14] J. Bolte, S. Sabach, and M. Teboulle. Proximal alternating linearized minimization for nonconvex and nonsmooth problems. *Mathematical Programming*, 146(1-2):459–494, 2014.

[CDDY06] E. Candes, L. Demanet, D. Donoho, and L. Ying. Fast discrete curvelet transforms. *Multiscale Modeling & Simulation*, 5(3):861–899, 2006.

[CDS98] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, 1998.

[CMTD14] O. Chabiron, F. Malgouyres, J.-Y. Tourneret, and N. Dobigeon. Toward fast transform learning. *International Journal of Computer Vision*, pages 1–22, 2014.

[CPR13] E. Chouzenoux, J.C. Pesquet, and A. Repetti. A block coordinate variable metric forward-backward algorithm. Technical Report 00945918, HAL, Dec. 2013.

[DCS09] J. M. Duarte-Carvajalino and G. Sapiro. Learning to sense sparse signals: Simultaneous sensing matrix and sparsifying dictionary optimization. *IEEE Trans. Image Process.*, 18(7):1395–1408, 2009.

[EAHH99] K. Engan, S. O. Aase, and J. Hakon Husoy. Method of optimal directions for frame design. In *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing (ICASSP)*, pages 2443–2446, Washington, DC, USA, 1999.

[Ela10] M. Elad. *Sparse and redundant representations: From theory to applications in signal and image processing*. Springer, 2010.

[Kin01] N. Kingsbury. Complex wavelets for shift invariant analysis and filtering of signals. *Applied and Computational Harmonic Analysis*, 10(3):234–253, 2001.

[Lan08] J. Landsberg. Geometry and the complexity of matrix multiplication. *Bulletin of the American Mathematical Society*, 45(2):247–284, 2008.

[LKF10] Y. LeCun, K. Kavukvuoglu, and C. Farabet. Convolutional networks and applications in vision. In *Proc. International Symposium on Circuits and Systems (ISCAS'10)*. IEEE, 2010.

[LMG14] L. Le Magoarou and R. Gribonval. Strategies to learn computationally efficient and compact dictionaries. In *International Traveling Workshop on Interactions between Sparse models and Technology (iTWIST)*, 2014.

[LMGG15] L. Le Magoarou, R. Gribonval, and A. Gramfort. FAμST: speeding up linear transforms for tractable inverse problems. In *Proc. European Signal Processing Conference (EUSIPCO)* , Nice, France, Aug. 2015.

[LS00] M. S. Lewicki and T. J. Sejnowski. Learning overcomplete representations. *Neural Computation*, 12(2):337–365, 2000.

[Mal99] S. Mallat. *A wavelet tour of signal processing*. Academic press, Boston, 1999.

[MBPS10] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *J. Mach. Learning Research*, 11:10–60, 2010.

[NT09] D. Needell and J. A. Tropp. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3):301–321, 2009.

[OF97] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, 37(23):3311 – 3325, 1997.

[PRK93] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*, pages 40–44. IEEE, 1993.

[RZE10] R. Rubinstein, M. Zibulevsky, and M. Elad. Double sparsity: Learning sparse dictionaries for sparse signal approximation. *IEEE Trans. Signal Process.*, 58(3):1553–1564, 2010.

[SO02] P. Sallee and B. A. Olshausen. Learning sparse multiscale image representations. *Advances in neural information processing systems*, pages 1327–1334, 2002.

[Str69] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[THZ13] T. Tsiligkaridis, A.O. Hero, and S. Zhou. On convergence properties of Kronecker graphical lasso algorithms. *IEEE Trans. Signal Process.*, 61(7):1743–1755, 2013.

[Tib96] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[TVW05] B. A. Turlach, W. N. Venables, and S. J. Wright. Simultaneous variable selection. *Technometrics*, 47(3):349–363, 2005.

[Wie12] A. Wiesel. Geodesic convexity and covariance estimation. *IEEE Trans. Signal Process.*, 60(12):6182–6189, 2012.